Code

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>

typedef struct {
    pthread_t thread_id;
    int thread_num;
    int sched_policy;
    int sched_priority;
} thread_info_t;

pthread_barrier_t barrier;
float time_wait = 0;

void busy_wait(float seconds) {
    clock_t start_time = clock();
    clock_t end_time = start_time + (clock_t) (seconds * CLOCKS_PER_SEC);
    while (clock() < end_time) {
    }
}

void *thread_fun(void *arg) {
    thread_info_t *thread_info = (thread_info_t *) arg;
    pthread_barrier_wait(&barrier);

    for (int i=0;i<3;i++) {
        printf("Thread %d is running\n", thread_info->thread_num);
        busy_wait(time_wait);
    }

    return NULL;
}

int main(int argc, char **argv) {
    int num_thread;
    char *policies;
    char *priorities;

    /* 1. Parse program arguments */
    int opt;
    while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1) {
        switch(opt) {
            case 'n':
                num_thread = atoi(optarg);
                break;
            case 't':
```

```
                time_wait = atof(optarg);
                break;
            case 's':
                policies = optarg;
                break;
            case 'p':
                priorities = optarg;
                break;
        }
    }

    int policy[num_thread];
    int priority[num_thread];

    char *p;
    p = strtok(policies, ",");
    for (int i=0;p!=NULL;i++) {
        if (!strcmp(p, "NORMAL"))
            policy[i] = SCHED_OTHER;
        else if (!strcmp(p, "FIFO"))
            policy[i] = SCHED_FIFO;
        p = strtok(NULL, ",");
    }

    p = strtok(priorities, ",");
    for (int i=0;p!=NULL;i++) {
        priority[i] = atoi(p);
        p = strtok(NULL, ",");
    }

    /* 2. Create <num_threads> worker threads */
    thread_info_t thread_info[num_thread];
    pthread_attr_t pthread_attr[num_thread];
    struct sched_param param[num_thread];
    pthread_barrier_init(&barrier, NULL, num_thread+1);

    /* 3. Set CPU affinity */
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(3, &cpuset);
    sched_setaffinity(getpid(), sizeof(cpuset), &cpuset);

    /* 4. Set the attributes to each thread */
    for (int i=0; i<num_thread; i++) {
        pthread_attr_init(&pthread_attr[i]);
        thread_info[i].thread_num = i;
        thread_info[i].sched_policy = policy[i];
        thread_info[i].sched_priority = priority[i];

        param[i].sched_priority = thread_info[i].sched_priority;
        pthread_attr_setinheritsched(&pthread_attr[i],
PTHREAD_EXPLICIT_SCHED);
        pthread_attr_setschedpolicy(&pthread_attr[i],
thread_info[i].sched_policy);
        pthread_attr_setschedparam(&pthread_attr[i], &param[i]);
```

```
        pthread_create(&thread_info[i].thread_id, &pthread_attr[i],
thread_fun, (void *)&thread_info[i]);
        pthread_setaffinity_np(thread_info[i].thread_id, sizeof(cpu_set_t),
&cpuset);
    }

    /* 5. Start all threads at once */
    pthread_barrier_wait(&barrier);


    /* 6. Wait for all threads to finish  */
    for (int i=0;i<num_thread;i++) {
        pthread_join(thread_info[i].thread_id, NULL);
    }

    pthread_barrier_destroy(&barrier);

    return 0;
}
```

Describe how you implemented the program in detail. (20%)

1. Parse program arguments

```
    int opt;
    while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1) {
        switch(opt) {
            case 'n':
                num_thread = atoi(optarg);
                break;
            case 't':
                time_wait = atof(optarg);
                break;
            case 's':
                policies = optarg;
                break;
            case 'p':
                priorities = optarg;
                break;
        }
    }

    int policy[num_thread];
    int priority[num_thread];

    char *p;
    p = strtok(policies, ",");
    for (int i=0;p!=NULL;i++) {
        if (!strcmp(p, "NORMAL"))
            policy[i] = SCHED_OTHER;
```

```
        else if (!strcmp(p, "FIFO"))
            policy[i] = SCHED_FIFO;
        p = strtok(NULL, ",");
    }

    p = strtok(priorities, ",");
    for (int i=0;p!=NULL;i++) {
        priority[i] = atoi(p);
        p = strtok(NULL, ",");
    }
```

利用`getopt()`取得參數後

- n的字串利用`atoi()`轉換成數字後存入`num_thread`
- t的字串利用`atoi()`轉換成數字後存入`timewait`
- s和p的字串分別先存入`char *policies`和`char *priorities`以便做`strtok()`分割字串的處理。
- 將`char *policies`和`char *priorities`利用`strtok()`將分割後的字串分別存入`int policy`和`int priority`中，便完成的parser資料前處理

2. Create <num_threads> worker threads

```
    thread_info_t thread_info[num_thread]; //
    pthread_attr_t pthread_attr[num_thread];
    struct sched_param param[num_thread];
    pthread_barrier_init(&barrier, NULL, num_thread+1);
```

此部分先用陣列將n個thread先行建立起來

- thread_info存放各個thread裡的pthread_t thread_id, int thread_num, int sched_policy, int sched_priority
- pthread_attr存放各個thread的屬性以便用於`pthread_attr_setschedpolicy()`, `pthread_attr_setinheritsched()`
- param裡存放各個thread的參數以便用於`pthread_attr_setschedparam()`
- `pthread_barrier_init()`啟用以便於之後讓各個process可以同時啟用,最後面的參數為所需等待thread呼叫`pthread_barrier_wait()`的數量

3. Set CPU affinity

```
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(3, &cpuset);
    sched_setaffinity(getpid(), sizeof(cpuset), &cpuset);
```

此部分啟用CPU affinity，這裡使用CPU 3，使得process將都在CPU 3做執行

4. Set the attributes to each thread

```

```

```
    for (int i=0; i<num_thread; i++) {
        pthread_attr_init(&pthread_attr[i]);
        thread_info[i].thread_num = i;
        thread_info[i].sched_policy = policy[i];
        thread_info[i].sched_priority = priority[i];
        param[i].sched_priority = thread_info[i].sched_priority;

        pthread_attr_setinheritsched(&pthread_attr[i],
PTHREAD_EXPLICIT_SCHED);
        pthread_attr_setschedpolicy(&pthread_attr[i],
thread_info[i].sched_policy);
        pthread_attr_setschedparam(&pthread_attr[i], &param[i]);
        pthread_create(&thread_info[i].thread_id, &pthread_attr[i],
thread_fun, (void *)&thread_info[i]);
        pthread_setaffinity_np(thread_info[i].thread_id, sizeof(cpu_set_t),
&cpuset);
    }
```

- 此部分將parse到的參數分別存入對應的thread_info和param中
- `PTHREAD_EXPLICIT_SCHED`會使用`pthread_create()`調用的屬性
- `pthread_attr_setschedpolicy()`設定schedule policy
- `pthread_attr_setschedparam()` 將param的參數存入attr
- `pthread_create()`會建立新的thread,第一個參數為thread, 第二個參數為thread相對應的attr, 第三個參數為所要執行的function,第四個參數為function的參數
- `pthread_setaffinity_np()`將設定CPU affinity,這裡用上面步驟相同的cpuset

5. Start all threads at once

```
    pthread_barrier_wait(&barrier);
```

```
void *thread_fun(void *arg) {
    thread_info_t *thread_info = (thread_info_t *) arg;
    pthread_barrier_wait(&barrier);

    for (int i=0;i<3;i++) {
        printf("Thread %d is running\n", thread_info->thread_num);
        busy_wait(time_wait);
    }

    return NULL;
}
```

- 呼叫`pthread_barrier_wait()`的數量到達`pthread_barrier_init()`的設定值後才開始執行

6. Wait for all threads to finish

```

```

```
    for (int i=0;i<num_thread;i++) {
        pthread_join(thread_info[i].thread_id, NULL);
    }

    pthread_barrier_destroy(&barrier);
```

- `pthread_join()`將等待thread結束
- `pthread_barrier_destroy()`destroy barrier並歸還資源

Describe the results of ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30 and what causes that. (10%)

```
Thread 2 is running
Thread 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
Thread 1 is running
Thread 0 is running
Thread 1 is running
Thread 1 is running
```

由於Linux採用CFS，所以不會保證realtime(FIFO)的process會比normal的process早跑，但是優先度較高的的FIFO一定會比優先度低的FIFO早跑，因此結果process 2一定會比process 1早跑

Describe the results of ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30, and what causes that. (10%)

```
Thread 3 is running
Thread 3 is running
Thread 3 is running
Thread 2 is running
Thread 0 is running
Thread 1 is running
Thread 1 is running
Thread 2 is running
Thread 0 is running
Thread 1 is running
Thread 0 is running
Thread 2 is running
```

由於Linux採用CFS，所以不會保證realtime(FIFO)的process會比normal的process早跑，但是優先度較高的的FIFO一定會比優先度低的FIFO早跑，因此結果process 3一定會比process 1早跑

Describe how did you implement n-second-busy-waiting?

```
void busy_wait(float seconds) {
    clock_t start_time = clock();
    clock_t end_time = start_time + (clock_t) (seconds * CLOCKS_PER_SEC);
    while (clock() < end_time) {
    }
}
```

利用clock_t先取得現在的時間當作start_time，設定好n秒後的end_time的時間，利用while迴圈去等待現在時間(clock())大於end_time的時間即可以做到busy waiting